Assisting the Visually Impaired Through Machine Learning Techniques

An Honors Thesis (HONR 499)

by

Tara Williams

Thesis Advisor

Dr. Shaoen Wu

Ball State University
Muncie, Indiana

May 2017

Expected Date of Graduation

May 2017

Abstract

In computer science today, machine learning presents itself as a rather broad and far-reaching field of study. Many previously unsolvable problems have found solutions through various machine learning techniques. The Internet provides large amounts of data that need efficient processing algorithms for use in practice. Machine learning techniques cut down on processing time for problems such as image classification, fraud detection, natural language processing, speech recognition, and so on. A convolutional neural network, one such machine learning solution, is one of the most accurate new methods for image recognition. This research experiment determines whether or not a convolutional neural network known as AlexNet can be used effectively in the context of assisting those with visual impairments. Small margins of error result in disaster if a person depends upon correct image recognition to navigate his or her environment. As such, the experiment tests ten types of images, and relevant statistics reveal whether or not the convolutional neural network is reliable enough for practical use.

Acknowledgments

I would like to thank Dr. Shaoen Wu for advising me during the course of this project and for helping me choose a topic that piqued my interest. Also, I would like to thank his graduate assistant, Junhong Wu, for assisting me and guiding me to research materials throughout my time working on this project.

Table of Contents

Process Analysis Statement

The field of computer science encompasses a wide range of disciplines and topics. Almost every business requires some form of computer science knowledge to function nowadays. So, during my junior year of college at Ball State University, I wondered what special topic I might find interesting. My classes offered different disciplines, yet I did not find anything particularly engaging. An early childhood interest in science and robotics led me to speak to a couple of professors, and I discovered machine learning. Instantly enthralled, I realized the potential in this field of work. Highly mathematical and scientific, it fit my interests and skillset.

Once I spoke to Dr. Wu, he recommended that I learn the basics of machine learning and research some current machine learning methods. So, I began my quest to understand how machine learning worked and, more importantly, why it worked. At first, the research was slow, difficult work. I learned from online machine learning courses, videos, and published articles. Machine learning encompasses complicated mathematical concepts, and though I had plenty of mathematical experience, translating written functions to programs takes a higher degree of skill. The concepts made sense, however, and I worked my way to a simple understanding.

Then, I needed to choose a specific machine learning topic to research. When Dr. Wu recommended it, given my previous experience with art, I thought an image recognition task suited me. As I stated before, I also enjoyed robotics, and image recognition programs are used to aid robot movement. Understanding convolutional neural networks, which are used for image classification, took quite a bit of time. I spent almost a month trying to grasp the concepts behind it. Finally, I coded a quick, short program that simulated a convolutional neural network. Simple and ineffective, it could not classify the most basic of images, yet it helped me gain a better understanding of the math behind the convolutional neural network. At times, programming concepts are best learned by implementing them, as theory only goes so far.

At last, I moved on to the programming project itself. The graduate student assigned to assist me in my project suggested that I implement AlexNet, a convolutional neural network created to solve the ImageNet problem. While I had wanted to create my own convolutional neural network, I realized throughout my research that programming an effective, accurate convolutional neural network on my own was simply not feasible; such a task takes professionals years to accomplish. Producing such a program would be exhaustive, even without the training step, which takes from weeks to months with processors much faster than my computer contains. If I had completed my own convolutional neural network, the results I would have obtained from it likely would have had little value.

As a result, I used AlexNet for my convolutional neural network, and I obtained open source classes and trained weights so I could skip the lengthy training process. My input to the program included static images taken from the Internet and live images I either took by camera or through my computer's video camera. I analyzed the data from these images and drew conclusions based on the results.

Throughout the course of this project, I learned quite a bit about research and my own interests. As a result of this project, I have decided to pursue a career in machine learning. Unlike some

areas of computer science, machine learning forces me to challenge myself and innovate creative solutions to problems. It holds my interest with its mathematical and scientific concepts. During this project, I also learned that a problem may seem simple on the outside yet be extremely complex when digging a little deeper. Image recognition looks so easy, but the actual problem involves variables that one would not even consider at first glance.

Overall, this project had its challenges and successes, and I found it an exciting change from my ordinary computer science courses. While not innovative or anything new, this project helped me discover my passion within computer science, and it holds great value to me because of this. I am thankful to those who guided and assisted me along the way, as I might never have found my true calling without this project.

I. INTRODUCTION

Accurate image classification is a difficult machine learning problem that plagues computer scientists today. Many new inventions, such as self-driving cars, facial recognition, and virtual reality rely upon image classification to work safely. Convolutional neural networks (CNNs) classify images with much better results than traditional supervised learning methods and produce accuracies above 99%. In this research paper, a convolutional neural network known as AlexNet is used to test both static images and live images, or images taken from a live video. It tests ten classes of images for both the live and static data sets, and each class contains ten images. So, the convolutional neural network tests a total of two hundred images. This project studies the results of the image tests in order to come to a meaningful conclusion about the usefulness of this particular AlexNet implementation in assisting those with visual impairments.

II. DEFINITIONS

A. Machine Learning

Machine learning is, in essence, a form of artificial intelligence. It encompasses a broad array of problems and tasks from image classification to speech recognition. Common issues in machine learning include facial recognition, fraud detection, and choosing ad popups based on a previous search history. It differs from regular programming in one essential way; most programming requires explicit commands for each and every step the program needs to perform, while machine learning gives the program tools it needs to learn and leaves out intermittent steps. For example, in image classification, a traditional program would be given an image of a cat and be told the image is a cat. On the other hand, an image classification program would be given example images of other cats from which to learn how to define a cat. From that collected data, it should be able to determine that the new image is a cat without the programmer telling it so directly.

B. Supervised Learning

There are three main types of machine learning: supervised learning, unsupervised learning, and reinforcement learning. Unsupervised learning works well with data sets that can be "clustered." These types of data sets usually have several groups of related data based on one or many features of the data. Supervised learning works best in classification problems when both the data and the class is known. In other words, unsupervised learning does not require beforehand knowledge of the classes assigned to the data, and supervised learning does [11]. This project uses supervised learning in order to utilize convolutional neural networks. As with the example used before, in order to classify an image of a cat in supervised learning, a large data set of different cat images labeled as cats will be used to train the classifier.

C. Image Classification

Image classification is a machine learning task of determining the class of an image based on a set of pre-trained classes. For example, if the program receives an image of a cat and has five classes—cat, dog, fish, goat, and horse—then the program should return a class of cat. Each class must be pre-trained, and programmers pre-train image classifiers with thousands to hundreds of thousands of images.

D. (Artificial) Neural Network

An artificial neural network (ANN), or just neural network (NN), is a computer program intended to mimic the human brain to gain processing power equal to or greater than that of a human's. In reality, the human brain cannot as of yet be replicated given its complexity and our lack of understanding of all its functions and intricacies. Each neural network tends to focus on a specific task, however, and uses a model similar to that of neurons in the brain, albeit with far less neurons, layers, and connections [18]. A multi-layer neural network, the most common and most effective neural network in image classification, includes at minimum three layers: an input layer, a hidden layer, and an output layer [15]. In the case of image classification, the input layer holds the raw images, usually with some preprocessing to get rid of outliers and normalize the data. The hidden layer determines important features of the data and adjusts its parameters based on those features [14]. An image classifier might decide to place importance on outlines or depths of the images. The output layer holds the final result of the neural network computations and can be converted into a class value.

E. Convolutional Neural Network

Convolutional neural networks are neural networks used for image recognition [13]. They are a more recent machine learning method used for image classification. CNNs perform well on image data sets, as they take into account the height, width, and depth of each image [13]. CNNs often involve multiple layers and alternate between convolutional and pooling layers. Convolutional layers take the input and pass a filter over it, creating a two-dimensional vector that indicates which points in the input share patterns [1]. Pooling layers reduce the size of the resulting vector from the convolutional layer in order to downsize the parameters of the image, making for faster computations [8]. Essentially, with each convolutional layer, small portions of the image are passed over many times, and obvious patterns between sections of the image present themselves. CNNs do not require pooling, but it is currently still in use.

F. Fully-Connected Neural Network

A fully-connected layer of a neural network is one where each neuron in the current layer connects to each neuron in the next layer. The neurons in the same layer do not connect to each other. If the first layer contains three neurons and the second contains four neurons, then twelve edges should connect the first and second layers. Fully-connected layers tend to be used after convolutional and pooling layers, and this project uses three fully-connected layers.

G. Deep Learning

Deep learning encompasses a wide range of machine learning problems and has led to faster computation time and more accurate results for common issues, such as image classification. Multi-layer artificial neural networks and convolutional neural networks are deep learning constructs. Deep learning utilizes multiple layers with varying degrees of abstraction in order to learn important information from the data it receives [1]. Deep learning methods exist for both supervised and unsupervised learning, and deep learning works well with large datasets [4]. Without deep learning, many datasets are too computationally expensive due to their size to train for various machine learning tasks. In this project, the convolutional neural network uses multiple

convolutional layers, pooling layers, and backpropagation to discover significant features about the images it attempts to classify.

H. Linear Regression

Linear regression determines whether or not a relationship exists between any number of variables [16]. Image classification techniques attempt to find the relationship between the image parameters (color, depth, outlines, etc.) and the class the image represents. As an example, a linear regression equation looking at a cat should discover that the pixels making up the image represent a class of cat. The simple form of the equation used for image classification is $f(W, b, x) = Wx + b$. W stands for weight, b stands for bias, and x typically represents the data from one image [9]. Most image classifiers set W, x, and b as vectors of varying dimensions. The linear regression equation may also be known as a linear classifier given that its output represents class scores.

I. Weights

When crafting a linear classifier, the weight vector is the most important variable. In a simple example, imagine W is a vector with dimensions [K x D], where K = the number of classes and D = the size of the image represented as a one-dimensional vector of pixels [9]. In this simple example, each row of W represents pixel values of one specific class. When multiplied by the image vector x, which is size [D x 1], this results in a vector of size [K x 1], where each row represents a single class value. The highest class value corresponds to the correct class for the image. Finding good initial values for the weights is vital for accurate classification, and trial and error testing is required to find the best weights. The weights also change during the training portion of the classifier and adjust constantly to better fit the data.

J. Bias

While not technically related to the image data, the bias vector keeps the weights from reaching zero. If the weight vector reaches zero, this prevents it from being adjusted. Following from the example above, the bias vector would have dimensions [K x 1], as it is added to the result of Wx [9]. Linear classifiers do not require a bias vector, but they perform better with them.

K. Loss Function

Loss functions attempt to minimize the "loss" or "risk" of a given problem [5]. A lower loss indicates a better classifier, and one would expect that a classifier with a very small loss classifies images correctly most of the time. Loss functions are also known as cost functions, and examples of loss functions include Multi-Class Support Vector Machine and softmax, the latter of which will be elaborated upon later [9]. Linear classifiers need loss functions to adjust their weights in a way that reduces the classification error.

L. Optimization

Optimization lowers the loss in a loss function, and it does this by optimizing the weights and bias in the case of a linear classifier [2]. During the linear classifier's training period, the weights increase or decrease in small increments determined by the optimization method. A common approach is to compute the gradient from one step to the next [10]. This optimization method compares the loss for the current weight and the weight moved by a miniscule amount in another direction. Then, it takes

the weight with the smaller loss and changes the weight to that value. This method is known as gradient descent.

M. Stochastic Gradient Descent

While gradient descent iterates through each example in the data set to calculate the next weights, stochastic gradient descent calculates the gradient based on one example from the data set [3]. While this takes less time to compute than normal gradient descent, and it can compute a fair estimate, stochastic gradient descent tends to be less accurate than gradient descent. Stochastic gradient descent is an optimization method that does not get used much on its own in practice.

N. Learning Rate

In gradient descent, the learning rate, also known as the step size, is the amount by which the weights are increased or decreased to test the loss function results. Smaller learning rates work better most of the time. Learning rates that are too small, however, take much longer to compute. Learning rates that are too large might miss a local maximum or minimum, and the loss will increase rather than decrease. Choosing a good learning rate is vital in gradient descent.

O. Softmax

The softmax function is a loss function that normalizes class scores resulting from the linear regression function and turns them into values between zero and one. The values must also add up to one [9]. The highest value indicates the correct class. For example, if the vector holding the class scores displays [0, 1, 0] after applying the softmax function, then the second value is the predicted class.

P. Backpropagation

Much like it sounds, backpropagation calculates the gradient of a function by working backwards from the last output to the input [12]. The chosen loss function determines the loss based on the expected output and the actual output for each neuron in the neural network. Backpropagation does this with partial derivatives [12]. At each step during backpropagation, the weights are adjusted according to the loss function.

Q. Rectified Linear Unit

Rectified Linear Unit (ReLU) is an activation function that is applied after the linear classification function. The class scores from the linear classification function have irregularities, and ReLU throws out class scores below zero by taking the max of the class score and zero [7]. This can be used in a binary way; if the result is zero, then the class is not activated, and if the result is anything else, the class is activated.

R. Training, Testing, and Classifying

Several steps make up supervised learning and neural networks. Two data sets are required: training data and test data. All classification techniques require training data, which should be independent of the test data. A sample of 10,000 data objects might be 90% training data and 10% test data, or 80% training data and 20% test data, so long as the two data sets do not overlap. Of course, data sets of such a small size would likely make a poor classifier. No test data should ever be used during the training step. The training step trains the classifier using the training data. In this step, the weights and bias adjust to better fit the training data. If a class of images is left out of the training data, then the classifier will

be unable to classify images of that type. For example, if no pictures of dogs exist in our training data, then the classifier cannot recognize a dog presented to it later. When the training step finishes, the variables and parameters of the neural network should not be adjusted again. During the test step, the test data is run through the neural network, and the results of the training step can be seen. If the classifier performs poorly on the test data, the programmer can go back and make adjustments to the training data set or the neural network to improve performance. If it performs well, the neural network can then be used as a classifier. Images can be input as necessary and their predicted classes output to the user. Errors in classification still exist at this stage, but the classifier should be trained well enough to have high accuracy, usually at least 99%.

S. TensorFlow

TensorFlow is an open-source library that makes the process of machine learning programming easier. It includes many built-in functions used in neural networks such as softmax, matrix multiplication, and max pooling. This project uses TensorFlow as its basis for building, training, and testing its neural network.

T. OpenCV

OpenCV is a library that this project uses to access the computer camera. It can open the camera, record frames, convert them to RGB or grayscale, resize the frames, and many other such functions.

U. AlexNet

AlexNet is a deep convolutional neural network created by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton to solve the ImageNet problem. ImageNet contains over one million images and one thousand classes on which to train the data [6]. This project utilizes AlexNet, as constructing a convolutional neural network from scratch that could provide useful results requires time and experience that exceeds the scope of this project.

III. SYSTEM DESIGN

In order to collect classification data using an AlexNet implementation, I coded a short program that takes images as input and produces class probabilities as output. The program takes in ten images from the computer's camera each time it runs and classifies the ten images separately. In such a short run time, each image or frame is essentially the same, but small variations matter a great deal in image classification. The program uses OpenCV for the image capture, and the code from this source [17] for the AlexNet implementation. The source also includes pre-trained weights, so the training step is skipped in this implementation. Instead, the program starts classifying images immediately. Training a classification algorithm from scratch can take hours to weeks with large datasets, so a pre-trained model gives good results when one wants to observe the outcomes of a proven classification technique rather than invent an entirely new technique. The program is coded in Python. The first five layers of the neural network in this implementation are convolutional layers, and the final three are fully-connected layers. Each layer has pre-trained weight and bias matrices associated with it, and the dimensions and values vary based on the AlexNet specifications for each layer, which have been optimized.

A. Pseudocode

Below, I have written some pseudocode to demonstrate the general functionality of the program. There is a single class, AlexNet, and the rest of the program functions through the *main.py* file. The program runs by calling *python main.py* in the terminal, and it has no GUI (graphical user interface).

As shown in the program, the AlexNet neural network is created through the *build()* function. It does most of the work in this program, creating the five convolutional and three fully connected layers, then returning the softmax values from the final layer. This result contains the predicted classes for the image that was input to the *build()* function.

```
def main():
    frames = getFrames()

    // Create new AlexNet object
    alexNet = AlexNet()

    // Build AlexNet
    // Run TensorFlow session with AlexNet input and frames
    // Print out frame and the first five classes computed by TensorFlow equation


def getFrames():
    // Requests live images from OpenCV camera
    // Sets images to size of 227 X 227 pixels, as is required by AlexNet
    // Returns a set of 10 image frames
```

```
class AlexNet():

    def loadWeights():
        // Loads pre-trained weights and biases as an array of matrices into AlexNet


    def getWeights():
        // Gets the pre-trained weights and biases


    def getWeight(identifier):
        // Gets a single weight matrix by its identifier in the matrix array


    def getBias(identifier):
        // Gets a single bias vector by its identifier in the matrix array


    // Builds the AlexNet classifier with the pre-trained weights and biases and is used for classification

    def build(x):
        // First convolutional layer
        conv1 = conv(x,
        getWeight("conv1"),
        getBias("conv1"), dimensions)

        // ReLU applied to first convolutional layer as the activation function
        relu1 = TensorFlow.relu(conv1)

        // Local response normalization and max pooling functions for first convolutional layer
```

```
// Second convolutional layer
conv2 = conv(maxpool1,
getWeight("conv2"),
getBias("conv2"), dimensions)

// ReLU
relu2 = TensorFlow.relu(conv2)

// Local response normalization and
max pooling functions for second
convolutional layer


// Third convolutional layer
conv3 = conv(maxpool2,
getWeight("conv3"),
getBias("conv3"), dimensions)

// ReLU
relu3 = TensorFlow.relu(conv3)


// Fourth convolutional layer
conv4 = conv(conv3,
getWeight("conv4"),
getBias("conv4"), dimensions)

// ReLU
relu4 = TensorFlow.relu(conv4)


// Fifth convolutional layer
conv5 = conv(conv4,
getWeight("conv5"),
getBias("conv5"), dimensions)

// ReLU
relu5 = TensorFlow.relu(conv5)

// Max pooling function for fifth
convolutional layer


// Sixth fully-connected layer
fc6 = connected(maxpool5,
getWeight("fc6"), getBias("fc6"))

// Seventh fully-connected layer
fc7 = connected(fc6,
getWeight("fc7"), getBias("fc7"))


// Eighth fully-connected layer
fc8 = connected(fc7,
getWeight("fc8"), getBias("fc8"))


// Returns the softmax value of the
final fully-connected layer, a matrix
of values between 0 and 1
return TensorFlow.softmax(fc8)
```

B. Sequence Diagram

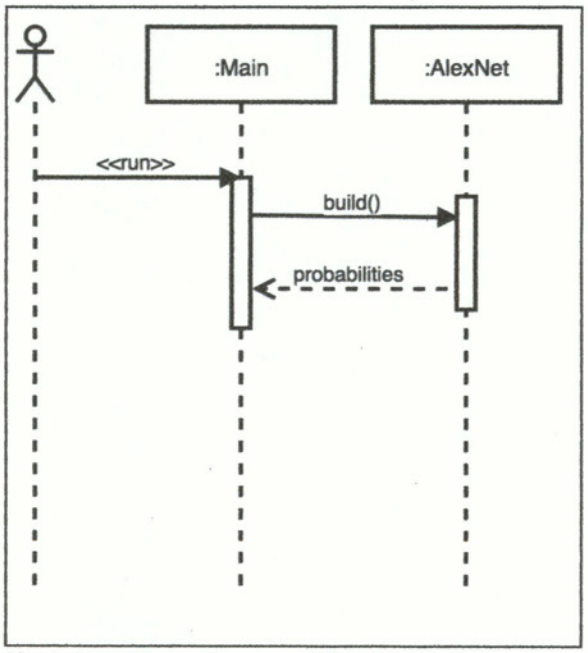

Fig. 1 A sequence diagram of the program.

The sequence diagram above shows the flow of the program. As one can see, the program is simple with few interactions between classes. This results from the heavy use of libraries in this project: OpenCV, TensorFlow, and Caffe classes do most of the work.

OpenCV contains all classes needed to open the webcam on the computer and receive the images. It also has functions to resize images, save images, and close the webcam. TensorFlow has all the necessary machine learning functions that are required to compute the class of each image. It has functions to compute the softmax values, rectified linear units, matrix multiplication, and so on.

Caffe classes and *bvlc_alexnet.npy* provide the classes and pre-trained weight data for the program to use. As such, these values do not have to be computed in the program itself.

In the sequence diagram, the user runs the program, and an AlexNet object is created. Then, main calls *build()* from AlexNet, which prompts AlexNet to run the convolutional neural network with the pre-trained weights and the frame to classify, producing the predicted classes for the frame. All other functionality happens through libraries and simple one-line functions.

In this diagram, one can see where issues might arise during the course of this experiment. By using libraries for the core functionality of the program, it may be difficult to determine the cause for some results found in the experiment. For example, the pre-trained weights make it challenging to determine whether an incorrectly classified image resulted from inadequate training data, as the training data is unknown. Additionally, the classes tested must be included in the Caffe classes list, which limits the number of classes that can be used in the experiment.

IV. EXPERIMENT

A. Static Images

In this experiment, data needed to be gathered from both the computer camera images and static images taken from the web. The static images provide a baseline for comparison, as AlexNet does not always perform perfectly, and static images contain fewer variables than live images. Ten classes of images were used to collect the static image data, and ten images were tested and classified for each class. Overall, one hundred static images provided a baseline to compare the live data against. The ten classes used were bicycle, soccer ball, car, book, shoe, chair, cat, dog, building, and traffic light.

Two key results determine the success of these classification tests: correct classification and probability. Correct classification is the percentage of images that are classified correctly in a given data set. For example, if eight out of ten images of soccer balls classify as soccer balls, then the correct classification is 80%. The classifier also determines a probability, and this is the likelihood that the image contains that class. For example, an image that has a 0.85 probability that the image contains a soccer ball indicates that the classifier is 85% certain it is a soccer ball. It might be 10% certain the image contains a golf ball, 5% certain the image contains a soft ball, and so on.

First, the static bicycle images tested rather well. The pre-trained classes contain different types of bicycle, so the bicycles sometimes tested as a different type of bicycle than their actual types. However, they all did classify as bicycles. The closest correct answer was 81.1% mountain bike.

Some factors appeared to affect the results, such as the angle at which the bicycle was positioned and whether or not the image included a background with the bicycle. The best results came from images of bicycles with no background that had a clear, straight side view of the bicycle.

Second, the soccer ball images tested well with some images and poorly with others. AlexNet seems to be trained with a typical black-and-white soccer ball. While the images tested well with or without backgrounds, they did surprisingly poorly when the soccer ball had a color scheme other than black and white. A blue soccer ball, for example, was classified as a wall clock. All but one example did recognize that it might be a soccer ball, however, and several listed it as the second or third most likely class.

The chart in Fig. 3 represents a correct classification of a soccer ball. This soccer ball looked like a traditional soccer ball, and it did not have a background.

Fig. 2 The soccer ball image used as a basis for the table in Fig. 3.

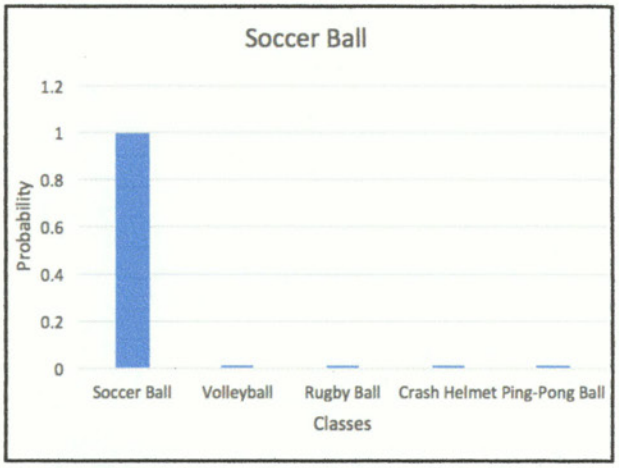

Fig. 3 A chart of one of the soccer ball images with its class probabilities.

As such, it classified correctly with a 99.4% chance that the image was a soccer ball. This probability measures up to the better classifiers that are currently being used in machine learning problems.

The chart in Fig. 5 represents an incorrect classification of a soccer ball. Instead, it chose "wall clock" as the most likely class at 38.7%. "Soccer ball" was chosen as the second most likely class, however, with a 24.5%.

Fig. 4 The soccer ball image used as a basis for the table in Fig. 5.

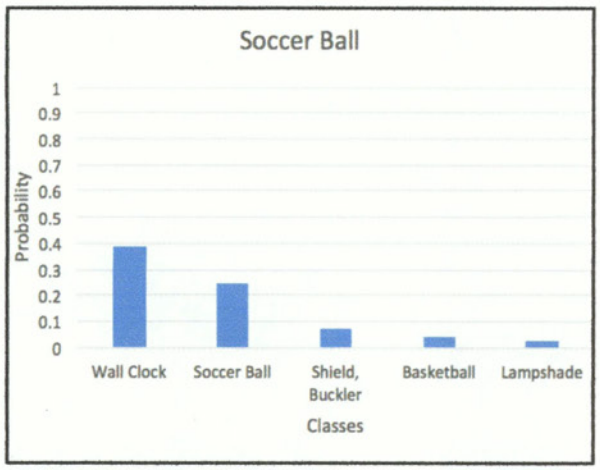

Fig. 5 A chart of a second soccer ball with its class probabilities.

The low probabilities in this case indicate an error either in the classification techniques or in the data set used to train the classifier. In this case, it seems more likely that the error originated in the training data.

Third, the car images classified with a high degree of accuracy. All were classified as some type of car. The probability of the car being a specific type of car tended to be low, however. This means that the classifier was certain that the image included a car, but it did not always choose the correct type of car. The probabilities ranged from 20% to 70%, though the second and third most likely classes were also types of cars. So, the classifier excels at classifying cars but sometimes fails to identify the type of the car.

The distribution of probabilities is also much more even than with something like a soccer ball, which has a single class in the class list. Since cars have multiple classes in the class list, individual probabilities tend to be lower even for correctly classified cars. For example, the car may be classified as 99% car, but just 30% station wagon.

Fig. 6 The car image used as a basis for the table in Fig. 7.

In Fig. 7, one can see the distribution more closely matches the incorrect classification of the soccer ball than it does the correct classification of the soccer ball. It does classify the image as a car, however, so the classifier should be more certain. Cars share a lot of features, so this makes the car more difficult to classify than a simply soccer ball. As such, the resulting probabilities reflect that difficulty. If one adds all car class probabilities together, the result is about 80%, twice the probability of the highest class prediction, 39.8%. It seems that more refined distinctions must be drawn when attempting to classify similar objects that share a large number of features.

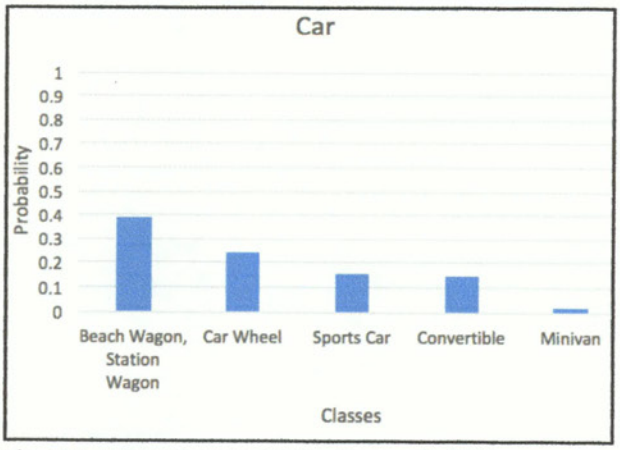

Fig. 7 A chart of a car with its class probabilities.

Books performed exceedingly poorly with this classifier. Though a couple of classes close to book existed, such as notebook or binder, it did not classify a single book as either of those classes. Instead, it chose a wide variety of seemingly unrelated classes such as quill, cucumber, printer, matchstick, and umbrella. The cause for such a huge error must be the training data used on this classifier. While a generic book class was not available, the books should have classified as binders or notebooks, as these classes resembled ordinary books. It seems unlikely that a cucumber shares more traits in common with a book than a notebook does.

Surmising that a notebook image might work better with this classifier, I tested one of those as well. While it did not classify correctly as a notebook, it did classify as a ring binder, but it still had a small probability. It appears that the training data did not have good examples of notebooks, or it did not have enough examples.

Fig. 8 The book image used as a basis for the table in Fig. 10.

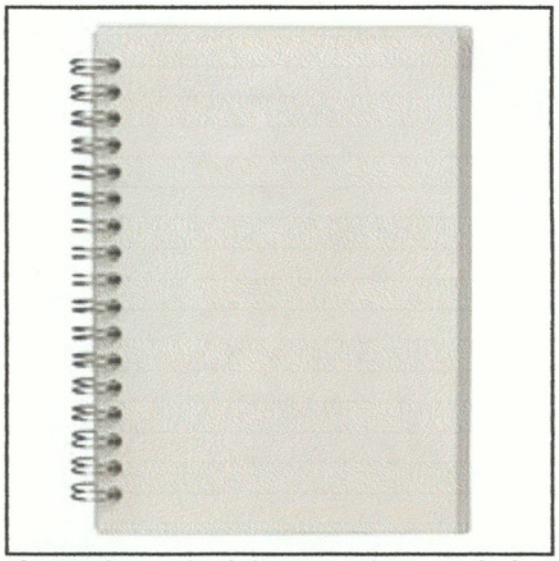

Fig. 9. The notebook image used as a basis for the table in Fig. 10.

For this class probability table, both the notebook and one of the books have been included to show the comparison between the two. Both are inaccurate, but the notebook has a slightly higher accuracy than the regular book. One can also see that the book image includes multiple books while the notebook image contains a single notebook, but single books also did not classify correctly in a consistent manner.

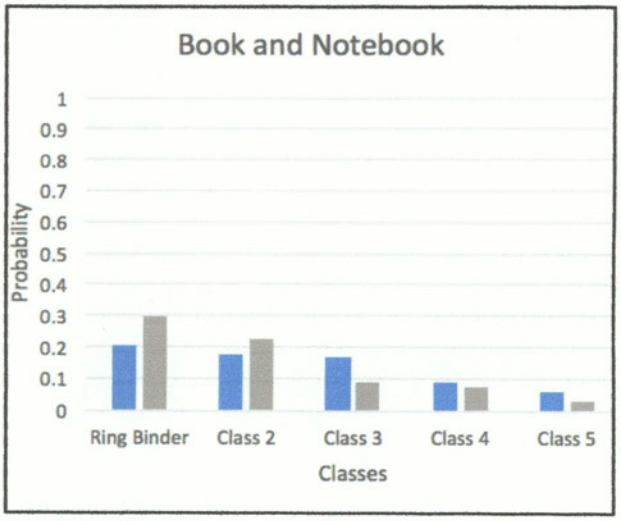

Fig. 10 A chart of a book and a notebook with their class probabilities. The blue bars represent the book, and the gray bars represent the notebook.

As shown in Fig. 10, the notebook and book classified as ring binders, even though a notebook class was included in the class list. While the notebook performs a bit better with this classifier, it still does not have a good probability with its predicted class at about 30%.

Next, shoes classified correctly 90% of the time. The classifier had difficulty with more than one shoe present in the image, but the issue seemed more prominent if the shoes were not the same color. Also, a greater error resulted from overlapping shoes. Two shoes that did not overlap tested as well as single shoes on their own.

In Fig. 14, three shoe image probabilities are displayed. The bars display in order of the images shown of the shoes. As one can see, the three shoes and their class probabilities are fairly consistent.

Fig. 11 The first shoe image used as a basis for the table in Fig. 14.

Fig. 12 The second shoe image used as a basis for the table in Fig. 14.

Though each shoe image classifies correctly, the first two images classify with a 99.2% and 96.6% probability while the third image classifies with a 28.6% probability. The determining factor appears to be the overlap in the shoes, though the lightness of the shoes in the third image and the light background might account for some of the error in this particular case. Either way, shoes have classified with the second highest accuracy of the classes tested so far.

Fig. 13 The third shoe image used as a basis for the table in Fig. 14.

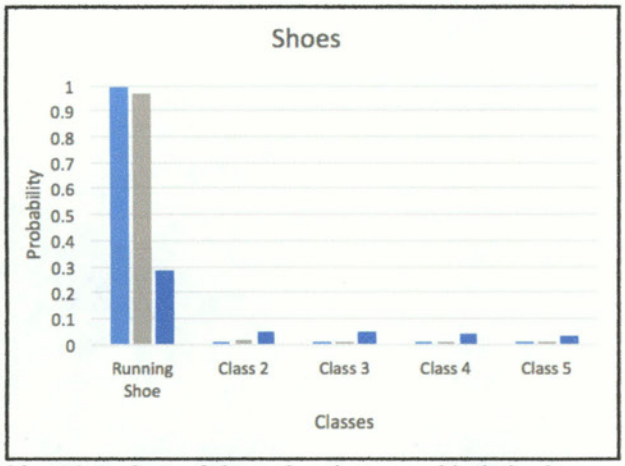

Fig. 14 A chart of three shoe images with their class probabilities. The light blue bars represent Fig. 11, the gray bars represent Fig. 12, and the dark blue bars represent Fig. 13.

The chair results varied greatly, from 0% to 99.8% probability that the image contained a chair. This likely reflects on the training data given its inconsistency. Padded chairs and armchairs were classified with low probabilities ranging from 0% to 39%, while typical dining chairs and folding chairs classified with high probabilities, from 76.8% to 99.8%.

Fig. 15 An example of a chair that classified with a high probability.

Fig. 16 An example of a chair that classified with a low probability.

The training data likely included more "dining room" chairs and less "comfortable" chairs. The average probability of the image being a chair over all ten images was about 42.3%, a low probability and the second lowest so far, the lowest being the books at 6.7%.

Next, the cats classified fairly accurately, and since the set of classes did not include a basic domestic cat, any class of cat is considered correct in this case. All but two of the images were recognized as cats. One image of a Siamese cat was identified correctly as a Siamese cat, and it was classified with the highest degree of certainty in the set of cat images at 74.2%. This indicates that the classifier identifies specific subspecies of cats accurately and makes guesses when the images contain other types of cats. Overall, 80% of the images were classified as cats.

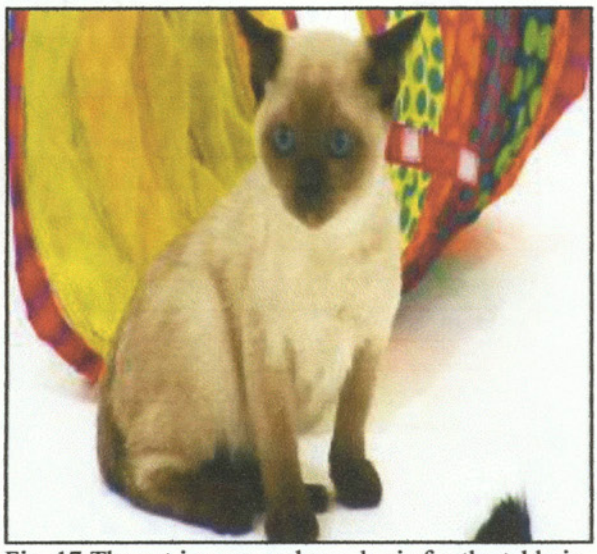

Fig. 17 The cat image used as a basis for the table in Fig. 18.

The Siamese cat is displayed in Fig. 17, and the bar chart of its class probabilities is shown in Fig. 18. Though it classified with a 74.2% probability, it should have a higher class probability given the clarity of the image and the lack of occlusion, or the lack of objects obscuring the cat, in the image. The other predicted classes were also not of cats but were instead dogs.

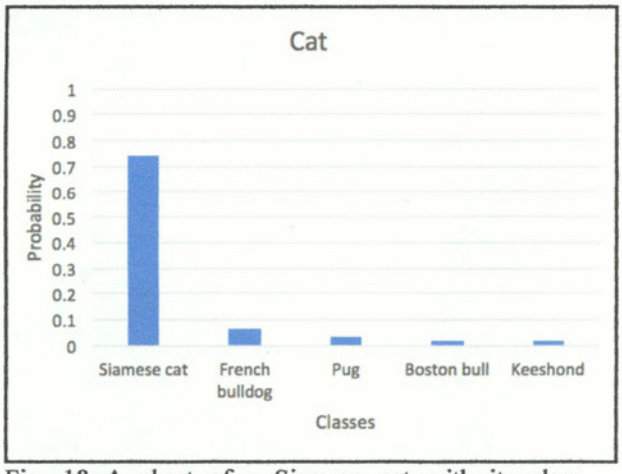

Fig. 18 A chart of a Siamese cat with its class probabilities.

Next, the program identified all but one of the dog images as dogs, but it tended not to choose the correct breed of dog. Of course, not all breeds are included in the classes, so it likely chose the closest breed to the pictured dog. It also had trouble when multiple dogs were in the image. A few of the dogs were classified as an incorrect breed when the correct breed existed in the list of classes. Factors such as the dog's posture and coat color seemed to matter in the classification of the images. Overall, 90% of the images classified as dogs.

Next, 40% of the building images classified as buildings. In this case, the classifier picked out specific objects in the image as opposed to selecting the building as a whole. The top class was picket fence, a common addition to houses. Tile roof also appeared several times as a probable class, and the buildings had tile roofs. The average probability that the images included buildings was about 48.1%.

Fig. 19 A traffic light with a probability of 88.8%.

Fig. 20 A traffic light with a probability of 26.0%.

Finally, traffic lights classified correctly 60% of the time. The error does not appear to be consistent, as similar traffic lights do not classify with the same probability. For example, two traffic lights with the same shape and close backgrounds classify with 88.8% and 26.0% probability.

A typical straight traffic light with no appendages classifies with the highest degree of accuracy. Double traffic lights or traffic lights with extra parts do not classify well with this training data set.

Fig. 21 summarizes the results of all the static image tests. Out of 100 images, 67% were classified correctly, and the average probability of the predicted class was 57.7%. These results are not favorable, and such a classifier cannot be used reliably with unpredictable results such as these. As such, live images will be tested to see if the results improve. Given the inconsistencies that present themselves in live video images, however, the results will likely worsen.

	Classified Correctly	Average Probability
Bicycle	1.0	0.867360007
Soccer Ball	0.6	0.596092781
Car	1.0	0.792435718
Book	0.1	0.067219847
Shoe	0.9	0.823152987
Chair	0.4	0.423187982
Cat	0.8	0.490493729
Dog	0.9	0.77035342
Building	0.4	0.481161277
Traffic Light	0.6	0.462438554
Average	0.67	0.57738963

Fig. 21 A table of each class tested, the correct classifications, and the average probabilities.

As one can see, the accuracy varied greatly for the ten classes tested in this experiment. Listed in order of correct classifications and highest average probability, the classes are: bicycle, car, shoe, dog, cat, soccer ball, traffic light, building, chair, book. The determining factor for accuracy seems to be the training data used to create the classifier. Specific images classified with upwards of 90% probability for each class, which indicates that the training data needed more variety to increase its accuracy.

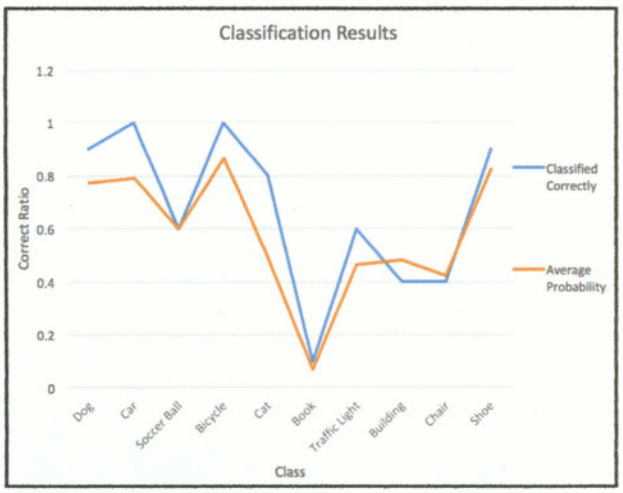

Fig. 22 A line chart displaying the results of each tested class.

B. Live Images

First, 100% of the bicycle images classified as bicycles. They classified as tandem bicycles, however, instead of mountain bicycles.

Fig. 23 An example of the bicycle classified from the live camera images.

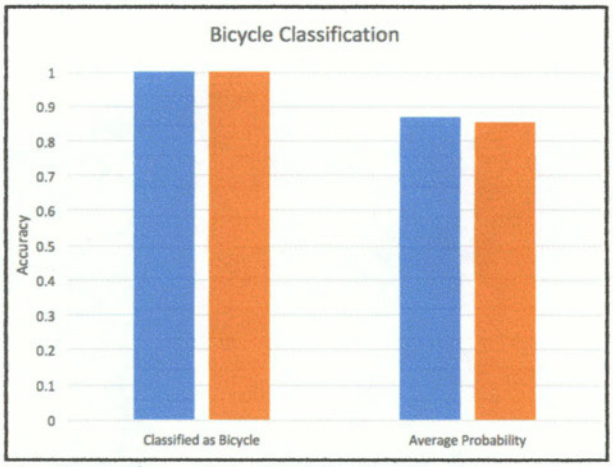

Fig. 24 A chart of the bicycle classification results. The blue bars represent the static images, and the orange bars represent the live images.

In Fig. 24, one can view the classification results of both the static and live images. Both had a 100% classification rate, which means all bicycles classified as some type of bicycle. Then, the average probability that the image was of a bicycle was 86.7% for the static images and 85.5% for the live images.

Both the static images and live images of the bicycles performed with a high degree of accuracy with this classifier. Though the probability of the image including a bicycle could be higher, only the first five classes were displayed in the classification results. The predicted class also tended to be the wrong type of bicycle, and backgrounds affected the outcome of the classifier. Overall, this classifier has a high enough degree of accuracy with bicycles to identify one reliably.

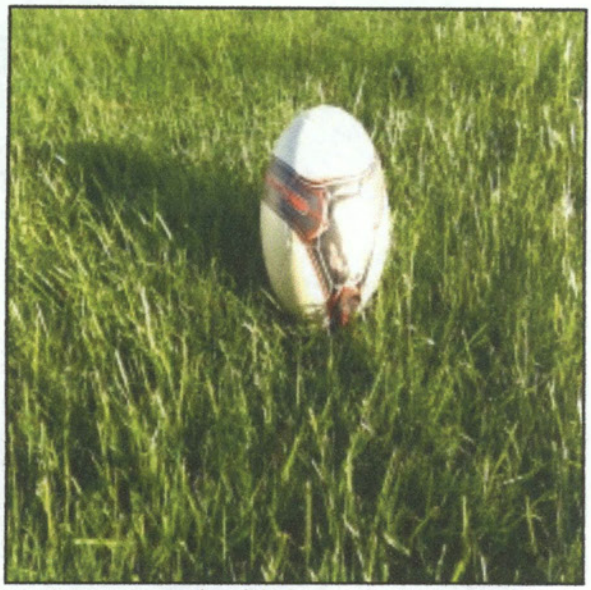

Fig. 25 An example of the soccer ball classified from the live camera images.

For the live soccer ball images, none classified as soccer balls. They classified as golf balls instead. The soccer ball was mostly white, which indicates that the classifier placed a great importance on the color of the object. In this case, the outline of the object needs a higher priority to classify the soccer ball accurately.

The static images had better results, but they included more traditional black-and-white soccer balls than the live images, which were all of the same soccer ball. While another soccer ball would have produced live results on par with those of the static images, testing the classifier on a non-traditional soccer ball has its uses. In this case, it shows the limitations of this particular classifier. The soccer ball has a clear outline, clear grooves, and little background. If the classifier cannot recognize a soccer ball in this case, then it cannot be relied upon to recognize a soccer ball in general.

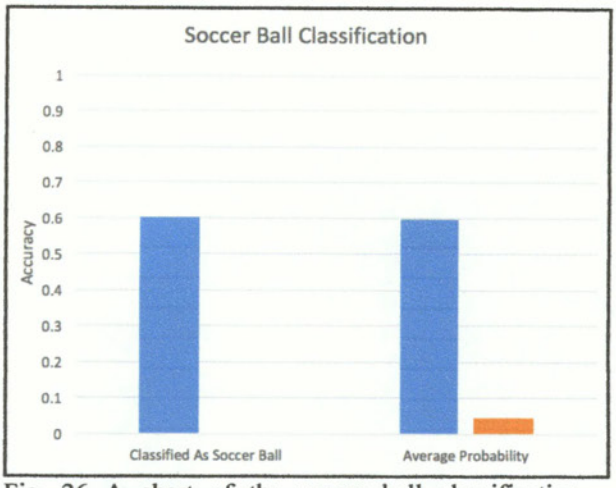

Fig. 26 A chart of the soccer ball classification results. The blue bars represent the static images, and the orange bars represent the live images.

As Fig. 26 shows, the accuracy of the classifier for soccer balls is 60% in the best case scenario and 0% otherwise. Overall, the classifier does not work well with soccer balls. It tends to choose a different type of ball, so it does not have enough information to distinguish between balls with small yet vital variations.

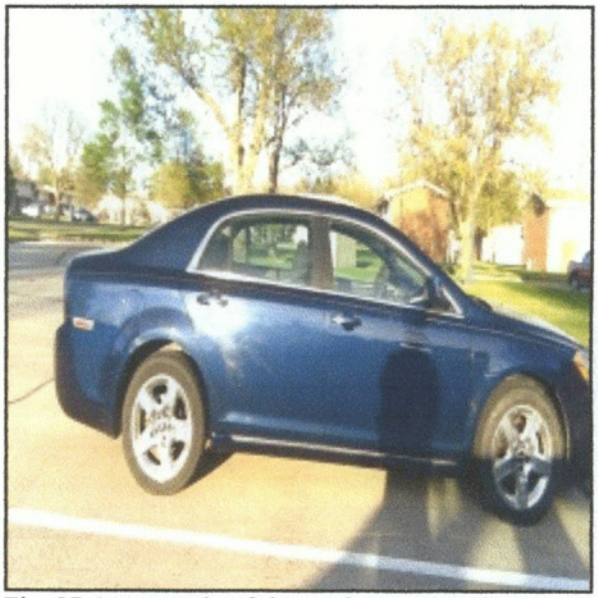

Fig. 27 An example of the car image classified from the live camera images.

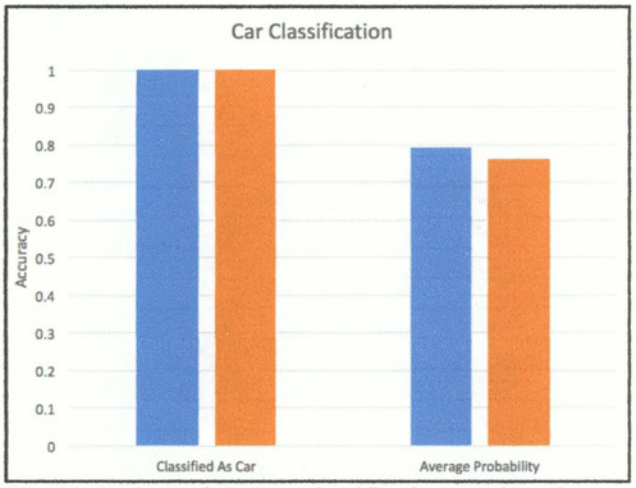

Fig. 28 A chart of the car classification results. The blue bars represent the static images, and the orange bars represent the live images.

Next, 100% of the live car images classified as cars with a 76.3% probability. This matches the static car image results.

Fig. 28 displays the results of both the static and live image tests. The static image classification performed slightly better, but the difference is negligible in this case. While less accurate than the bicycle classification, cars classify well with this classifier. This applies to cars viewed from the side; cars did not classify accurately if the image contained a direct frontal view of the car. Instead, the classifier tended to pick up the grille of the car and chose that as the top class as opposed to the entire car.

Books classified with a low degree of accuracy on all fronts. The chart of the classification results is not much use in this case, so it is not included in this report. Both the static and live images tested near 0% accuracy. For the most part, the reason for this seems to be that a generic book class is not available in this data set. However, one would expect books to classify as notebooks or binders, and they rarely did. In fact, the live images tested as memorial plaques or crossword puzzles. In all likelihood, the training data had a dearth of notebook and book images, so the books input to the classifier had little with which to compare. With this classifier, books are unreliable.

While the live shoe images classified correctly 100% of the time, they tested with a lower average probability than the static shoe images that classified correctly 90% of the time. The live shoe images all contained a single shoe as well and so did not have the issue of classifying multiple shoes.

Like bicycles and cars, the shoes classify with a reliable degree of accuracy. Whether live or static images, the determining factor seems to be the quality of the image and whether or not other objects overlapped the shoe. Distance also made a difference; live images taken from a distance of several meters almost always picked up other objects in the image to classify instead of the single shoe.

Fig. 29 An example of the shoe image classified from the live camera images.

Fig. 30 A chart of the shoe classification results. The blue bars represent the static images, and the orange bars represent the live images.

The live images of a chair classified correctly 100% of the time. For the live images, a single folding chair was used, unlike the greater variety of chairs used for the static images. The static images classified correctly 40% of the time, less than half the accuracy of the live images. The chair images perform well with folding chairs, but other chairs are inconsistent despite their similarities to folding chairs.

Fig. 31 An example of the chair image classified from the live camera images.

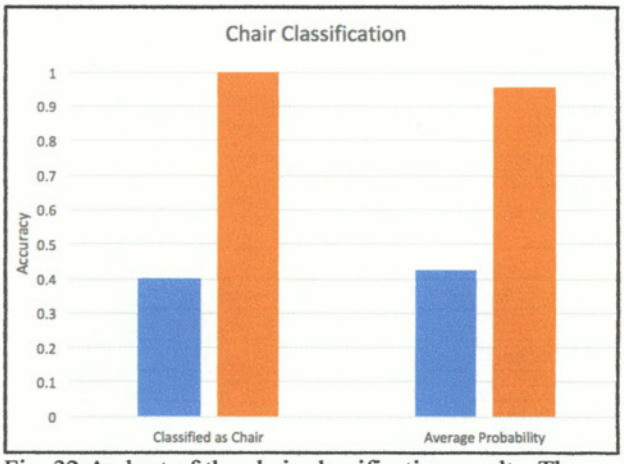

Fig. 32 A chart of the chair classification results. The blue bars represent the static images, and the orange bars represent the live images.

Fig. 32 shows the chair classification results for both the static and live images. In this case, folding chairs can be classified reliably, but other types of chairs cannot be classified as easily.

For the live cat images, I did not have a live cat on which to test my classifier, so I used pictures of my own cats. Only 40% of the images classified as cats, though the range of poses of the cats varied more than those in the static images.

Fig. 33 An example of a cat that did not classify correctly.

Fig. 34 An example of a cat that classified correctly.

As evidenced by Fig. 33 and Fig. 34, the cat's silhouette matters in the classification. Fig. 33 does not have a clear outline of the cat, as it is stretched out. Fig 34. has a clear outline with obvious pointy cat ears and cat features in the face. Though the whole cat is not visible in Fig. 34, it appears the cat's head needs to be fully visible for the classifier to recognize the cat.

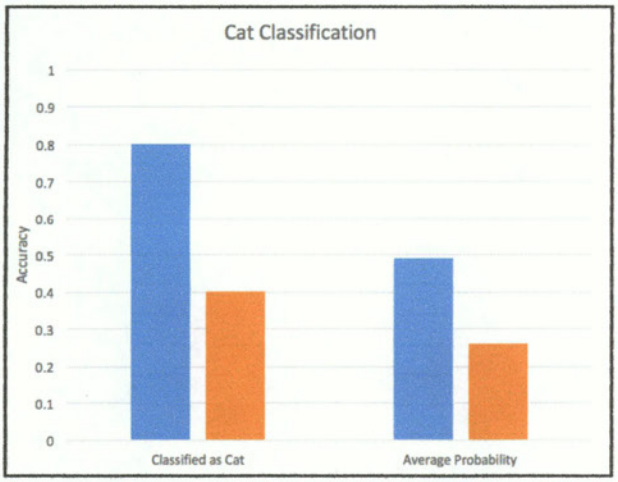

Fig. 35 A chart of the cat classification results. The blue bars represent the static images, and the orange bars represent the live images.

Overall, cats cannot be classified accurately. While the cats can be correctly classified a majority of the time with specific cat pictures, it does not apply to all cat pictures, and the average probabilities are quite low even when the cats classify correctly. The static images classify with an average 49.0% probability at best and 25.9% at worst. So, the classifier is only about 50% certain that the image contains a cat, which is not a good probability for a classifier such as this.

For the live dog images, I did not have a live dog on which to test my classifier, so I used pictures of my own dogs. All but one classified as a dog, and they classified as the correct breed most of the time. Two of the included dogs were mutts, but their predicted classes match their partial breeds. The one purebred dog in the image set was classified with a low probability despite being classified correctly.

Fig. 36 The dog image used as a basis for Fig. 37.

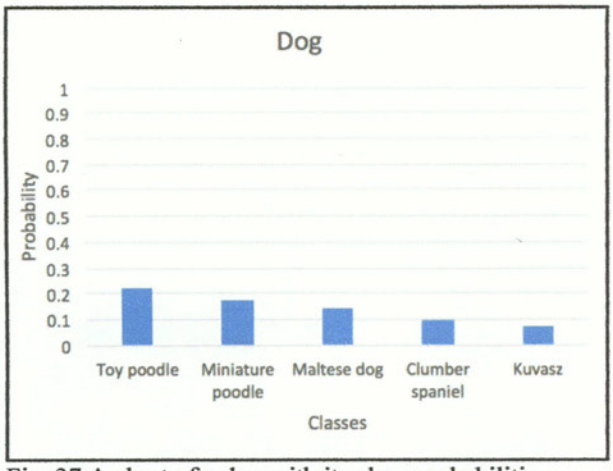

Fig. 37 A chart of a dog with its class probabilities.

The purebred, a toy poodle, classified correctly, but it has a fairly even distribution of predicted classes, as can be seen in Fig. 37. So, it seems the differences between breeds of dogs can be small, and similar breeds of dogs are more complicated to classify.

In Fig. 38, one can view the classification results of both the static and live images. Both had a 90% classification rate, meaning nine out of ten images were classified as some type of dog for each data set.

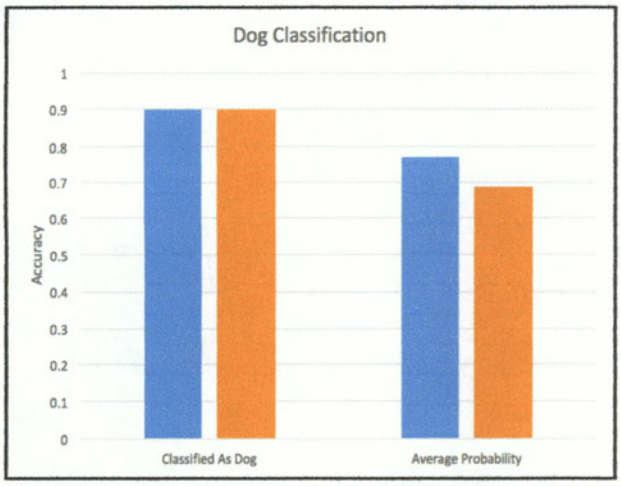

Fig. 38 A chart of the the dog classification results. The blue bars represent the static images, and the orange bars represent the live images.

Then, the average probability that the image was of a dog was 77.0% for the static images and 68.8% for the live images. This performance is very poor, as good classification algorithms can reach 99% accuracy. The classifier did not include all breeds of dogs, which may be one reason for the poor performance. In fact, when the two images of a dog without a breed in the class list (a dachshund) are thrown out of the live image set, the average probability jumps from 68.8% to 77.5%, an almost 10% increase. While the average probability still could be much better, it shows that the non-included dogs do affect the final results.

Other variables affect the classification results, such as the dog's posture, background elements, and overall photo quality. Higher resolution photos classified better than lower resolution photos. Another factor is the small data set, though in classification, while the training data set needs to be large, images input to the classifier after training should be classified correctly regardless of the size of the input.

Fig. 39 An example of a building image classified from the live camera images.

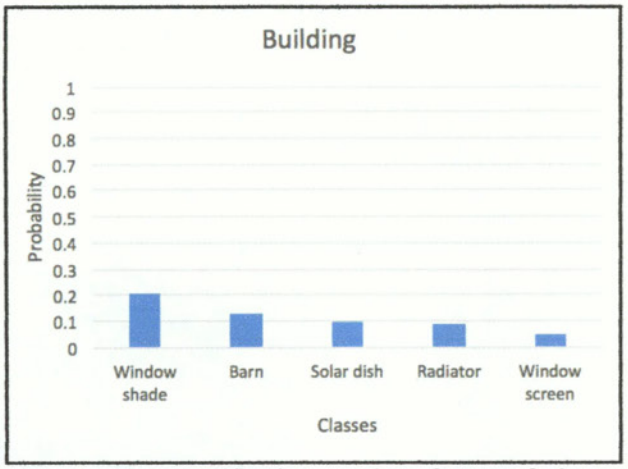

Fig. 40 The classification results of one of the buildings and the corresponding low probabilities.

The live building images used did not classify correctly most of the time. While all types of buildings are considered a success in this case, it still recognized the image as a building just 50% of the time. Given the rows of long, straight sections separating the windows, it seems the classifier commonly mistook the building as a radiator or a window shade. Despite the clear outline of the building, the classifier does not take scale or size into account. Fig. 40 displays the classification results of one of the building images. The highest probability is 20%, and it is an incorrect classification.

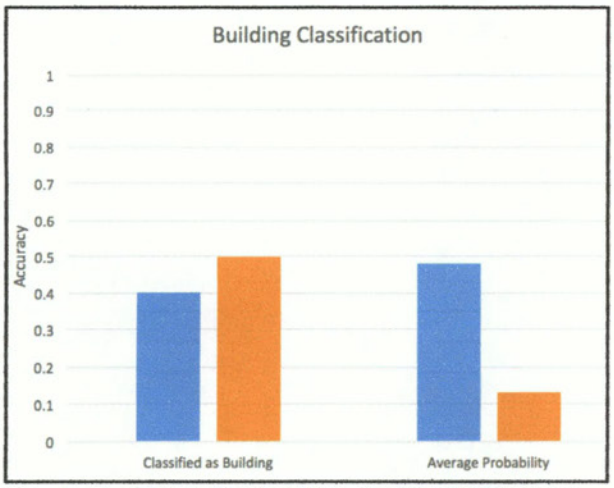

Fig. 41 A chart of the building classification results. The blue bars represent the static images, and the orange bars represent the live images.

The static images classified as buildings 40% of the time, but the average probability was more than three times that of the live images at 48.1%. Overall, given the results of these tests, buildings cannot be classified reliably with this classifier. It seems that, given the size of buildings and the distance from which they need to be photographed, the classifier too often recognizes other objects in the image, smaller objects or ones closer to the camera.

While traffic lights classified consistently for both static and live images, they classified with a low accuracy. 60% of the static images classified correctly, and 40% of the live images classified correctly. As taking live video of the traffic lights was dangerous, I took photographs of traffic lights instead.

Fig. 42 An example of a traffic light that classified correctly with a probability of 95.8%.

Fig. 43 An example of a traffic light that classified incorrectly with a probability of 13.7%.

Though all images contain a traditional traffic light with no appendages, some of the images have background objects obscuring the full outline of the traffic light. Fig. 43 classified incorrectly as a wallet. Another image of a partially obscured traffic light classified as a binder. For traffic lights, it seems that the classifier requires a clear outline in order to classify correctly.

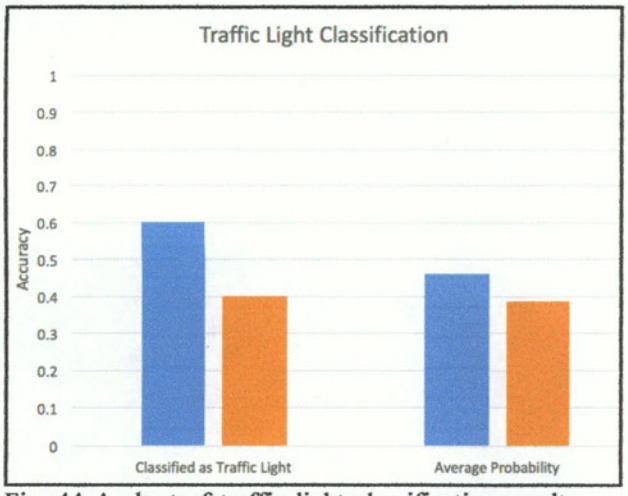

Fig. 44 A chart of traffic light classification results. The blue bars represent the static images, and the orange bars represent the live images.

	Classified Correctly	Average Probability
Bicycle	1.0	0.855429115
Soccer Ball	0.0	0.042972238
Car	1.0	0.762832119
Book	0.0	0.000000000
Shoe	1.0	0.807883225
Chair	1.0	0.954374999
Cat	0.4	0.258682879
Dog	0.9	0.68841547
Building	0.5	0.133647954
Traffic Light	0.4	0.388420587
Average	0.62	0.489265859

Fig. 45 A table of each class tested, the correct classifications, and the average probabilities.

All in all, the live images tested on par with the static images. Additional variables in the live image data set caused it to have lower classification accuracies and probabilities on average. Fig. 45 displays the results of the live image tests for each of the ten classes in this experiment. Listed in order of correct classifications and highest average probability, the classes are: chair, bicycle, shoe, car, dog, building, traffic light, cat, soccer ball, book. These results deviate from the static image results. On average, the static images are more accurate than the live images.

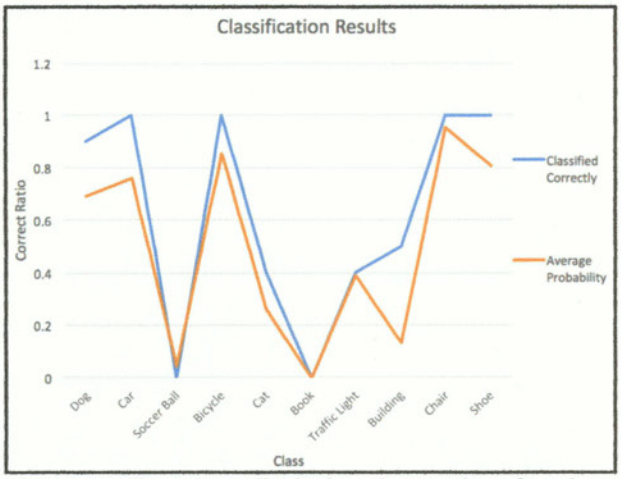

Fig. 46 A line chart displaying the results of each tested class.

Fig. 47 A line chart displaying the correct classification rate of each class tested. The blue line represents the static images, and the orange line represents the live images.

V. RESULTS

This project seeks to determine whether a classifier such as AlexNet can be used to aid a person with a visual impairment. Given the results found in these experiments, the answer is inconclusive. While some classes test with nearly perfect accuracies and probabilities, others perform at an almost 0% correct rate. Much can be learned from this convolutional neural network, however, and its practical applicability depends upon the training data used to train it.

One conclusion drawn from these experiments is that several classes have been trained well enough for practical use. Bicycles, cars, shoes, and dogs classify correctly both in the static and live image data sets at least 90% of the time. The average probabilities are also above 68% for each of the four classes. The consistency in these cases indicates a good classifier. Fig. 47 and Fig. 48 show this.

The figures also show the inconsistency with the other classes tested in this experiment. Spikes in the graph indicate where the classes diverge, which should not happen with an accurate classifier. Soccer balls classified with a low accuracy if they did not look like traditional soccer balls. Chairs classified correctly if they matched a specific type of chair. Cats did not classify well because some breeds of cat were included in the class lists, but a generic cat was not included as a class. Books classified correctly almost 0% of the time given the lack of training data on books. Various reasons caused objects to classify incorrectly, but the common variable was a lack of training data similar to the test images.

Other variables that caused incorrect results included lighting, distance, other objects being detected, occlusion of the object, and the class either not being listed or not matching the test object despite similarities.

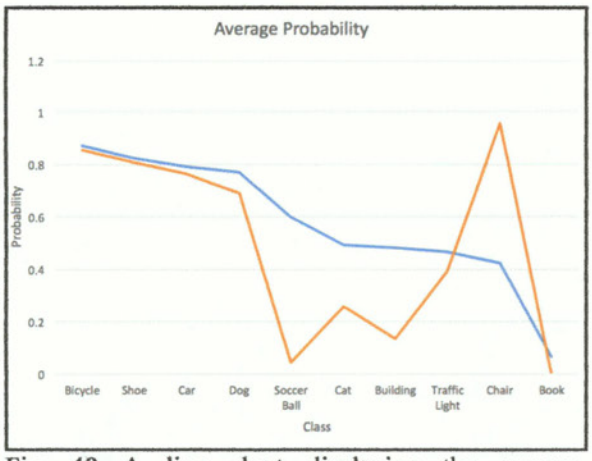

Fig. 48 A line chart displaying the average probabilities of each class tested. The blue line represents the static images, and the orange line represents the live images.

Fig. 49 shows the average percentage of images classified correctly and the average probability for each data set. The static image results are slightly better than the live image results, but overall, it is consistent. For both live and static images, the percent of correctly classified images is slightly above 60%.

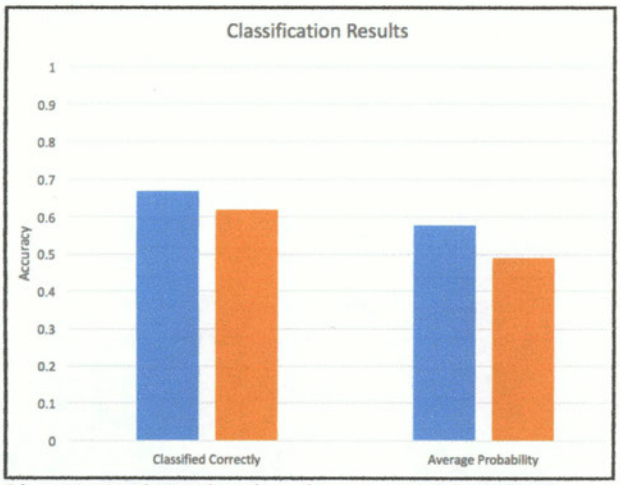

Fig. 49 A chart showing the average classification and average probability for each image set. The blue bars represent the static images, and the orange bars represent the live images.

The average probability that the image contained the predicted class was in the range of 50-60% for the static and live image sets. When taken together, this means that 60% of the objects classify correctly, and the classifier is 60% sure that the item contains the object. With simple probability, the total correctness of the program is just 0.6 X 0.6 = 0.36, or 36%. The other 64% represents either incorrect classification or uncertainty on the part of the classifier.

VI. CONCLUSION

While AlexNet is a promising convolutional neural network, this experiment shows that it does not always perform well with certain data sets. Whether this results from the training data used to train the classifier, the lack of certain objects in the class list, the unpredictability and multiple objects common in live image data sets, or some unknown factor, it seems this classifier needs some tweaks in order to perform with an acceptable degree of reliability, somewhere around 99%. Compared to the results of this experiment, this requires a jump in accuracy of about 30%. As said before, specific images most certainly would always classify near 99%. Clean images with no backgrounds and definite object outlines almost always perform well. The real world has a wide variety of variables, and the classifier should reflect this. A classifier that works in a best-case scenario has some use, but its practical applications are limited. In order to guide the visually impaired, the classifier cannot make mistakes, and if it does, those mistakes must be minimized.

If given a chance to do this project again with the knowledge I have now, I would add several steps to my analysis process. First, instead of using a single convolutional neural network as a classifier, several different classifiers would provide more diverse results. By using several classifiers, they could be compared in their accuracy and reliability. Then, I would test a larger data set for each classifier. With limited time, two hundred images were tested in this case, yet a larger set might provide different results. Additionally, with more time, I would have trained the classifier using my own training data set. This way, instead of using unknown training data, I would know exactly what data created the classifier and how it affected the classification results of the test data. Finally, I would have people test the program live instead of merely inputting images and calculating statistics from their classification results. Adding all of these elements might serve to better this project, though these aspects I recognized through hindsight and will consider in my future endeavors.

Overall, this experiment does not yield promising results. A visually impaired person could not travel safely using this specific classifier and its associated training data. The results from this project do provide some insight into common image classification problems, such as occlusion, background noise, lack of relevant training data, and similar yet different objects. Another lesson learned from this is the importance of choosing diverse test data to determine the accuracy of a classifier. If one chooses the same type of object for every example, it may give an overinflated accuracy. For example, folding chairs tested with over 90% accuracy with this classifier, yet similar chairs tested with less than half that accuracy. A lack of diverse test data leads to skewed results as much as a lack of training data does.

In conclusion, this classifier cannot be used to guide those with visual impairments. It did well with some objects but not with others, so its unreliability makes it unsuitable for navigation. Still, it must be acknowledged that, with another training data set, this classifier might yield better results. All in all, this experiment shows how unpredictable classifiers can be with certain data sets.

Works Cited

[1] Y. Bengio, G. Hinton, and Y. LeCunn, "Deep learning," *Nature*, vol. 521, pp. 436-444, May 2015.

[2] K. P. Bennett and E. Parrado-Hernandez, "The interplay of optimization and machine learning research," Journal of Machine Learning Research, vol. 7, pp. 1265-1281, July 2006.

[3] L. Bottou, "Large-scale machine learning with stochastic gradient descent," NEC Labs America, Princeton, NJ, 2010.

[4] L. Deng and D. Yu, "Introduction" in *Deep Learning Methods and Applications*. Redmond, WA. 2013, ch. 1, pp. 198-202.

[5] E. DeVito, A. Caponnetto, M. Piana, L. Rosasco, and A. Verri, "Are loss functions all the same?" University of Geneva, 2003.

[6] G. E. Hinton, A. Krizhevsky, and I. Sutskever, "ImageNet classification with deep convolutional neural networks," University of Toronto, Toronto.

[7] G. E. Hinton and V. Nair, "Rectified linear units improve restricted boltzmann machines," University of Toronto, Toronto.

[8] A. Karpathy. *Convolutional neural networks (cnns / convnets)* [Online]. Available: http://cs231n.github.io/convolutional-networks/#pool

[9] A. Karpathy. *Linear classification* [Online]. Available: http://cs231n.github.io/linear-classify/

[10] A. Karpathy. Optimization [Online]. Available: http://cs231n.github.io/optimization-1/

[11] P. Lison. (2012, October 3). *An introduction to machine learning* [Online]. Available: http://folk.uio.no/plison/pdfs/talks/machinelearning.pdf

[12] M. Nielsen (2017). *How the backpropagation algorithm works* [Online]. Available: http://neuralnetworksanddeeplearning.com/chap2.html

[13] J. C. Platt, P. Y. Simard, and D. Steinkraus. "Best practices for convolutional neural networks applied to visual document analysis," Microsoft Research, Redmond WA, 2003.

[14] D. A. Pomerleau and D. S. Touretzky. "What's hidden in the hidden layers?" Byte, pp. 227-233, Aug. 1989.

[15] S. Raschka. "Training artificial neural networks for image recognition," in *Python Machine Learning*. Birmingham, UK: Packt Publishing Ltd, 2015, ch. 12, pp. 341-385.

[16] X. G Su and X. Yan. *Introduction* in Linear Regression Analysis, Hackensack, NJ: World Scientific Publishing Co., 2009, ch. 1, pp. 1-5.

[17] *AlexNet implementation + weights in tensorflow* [Online]. Available: http://www.cs.toronto.edu/~guerzhoy/tf_alexnet/

[18] *A basic introduction to neural networks* [Online]. Available: http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html